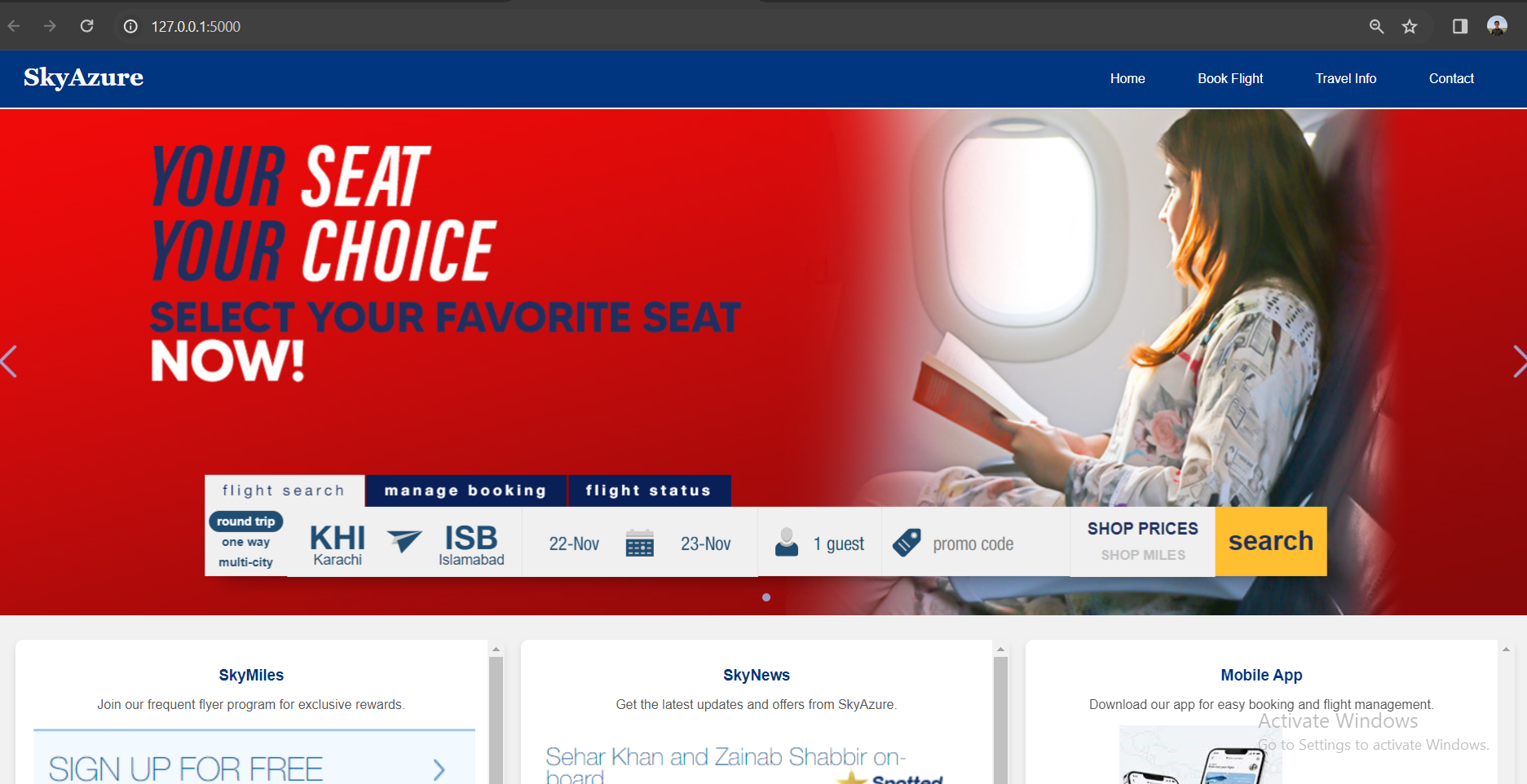
# **Data Pre-Processing**

We are given a csv file for each item, i.e., aircraft, airport and weather. We have written a class for each type, and a manager class which stores the loaded data into lists and also converts them into tables for simpler indexing.

# **Simulation**

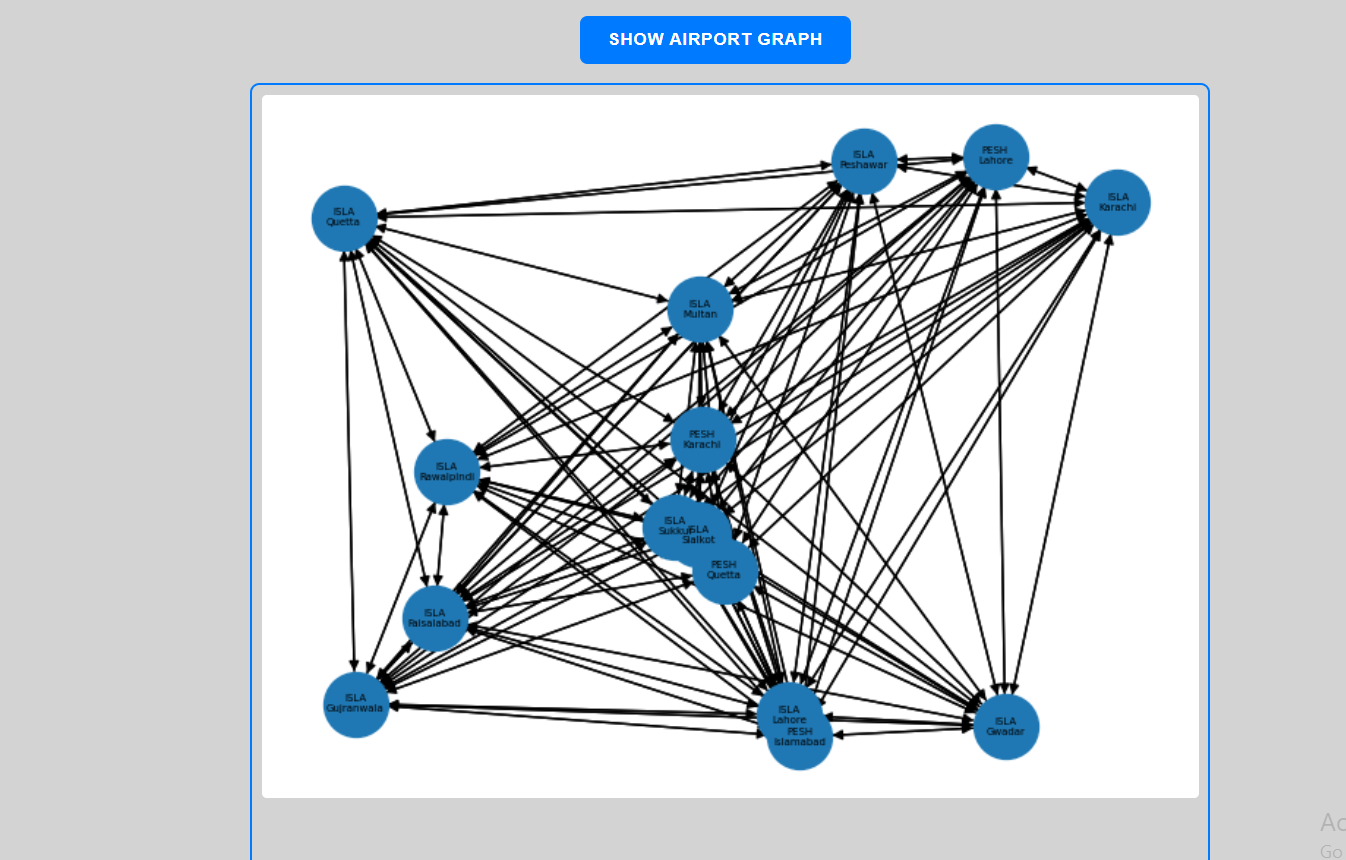
On the backend, the simulation is handled by the genetic algorithm, i.e., the selection of parents, crossover to create children, random mutation of the children and forwarding them to the next generation.

The front end was entirely based on html, css, javascript and flask establishing the connection between the Application Layer and the backend Layer. By using the html and the css an attractive interface was made. A hypothetical Flying company front end page was made for the admin side. The company was given the name SkyAzure. Here are the front end pages:  


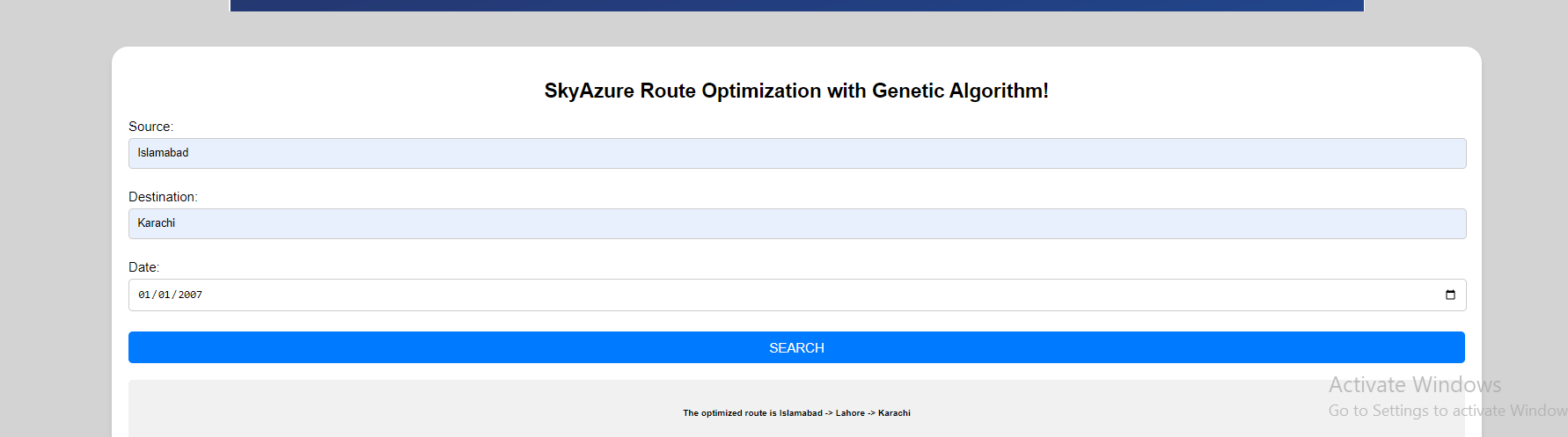
A screenshot of a computer

Description automatically generated  
  
A screenshot of a computer

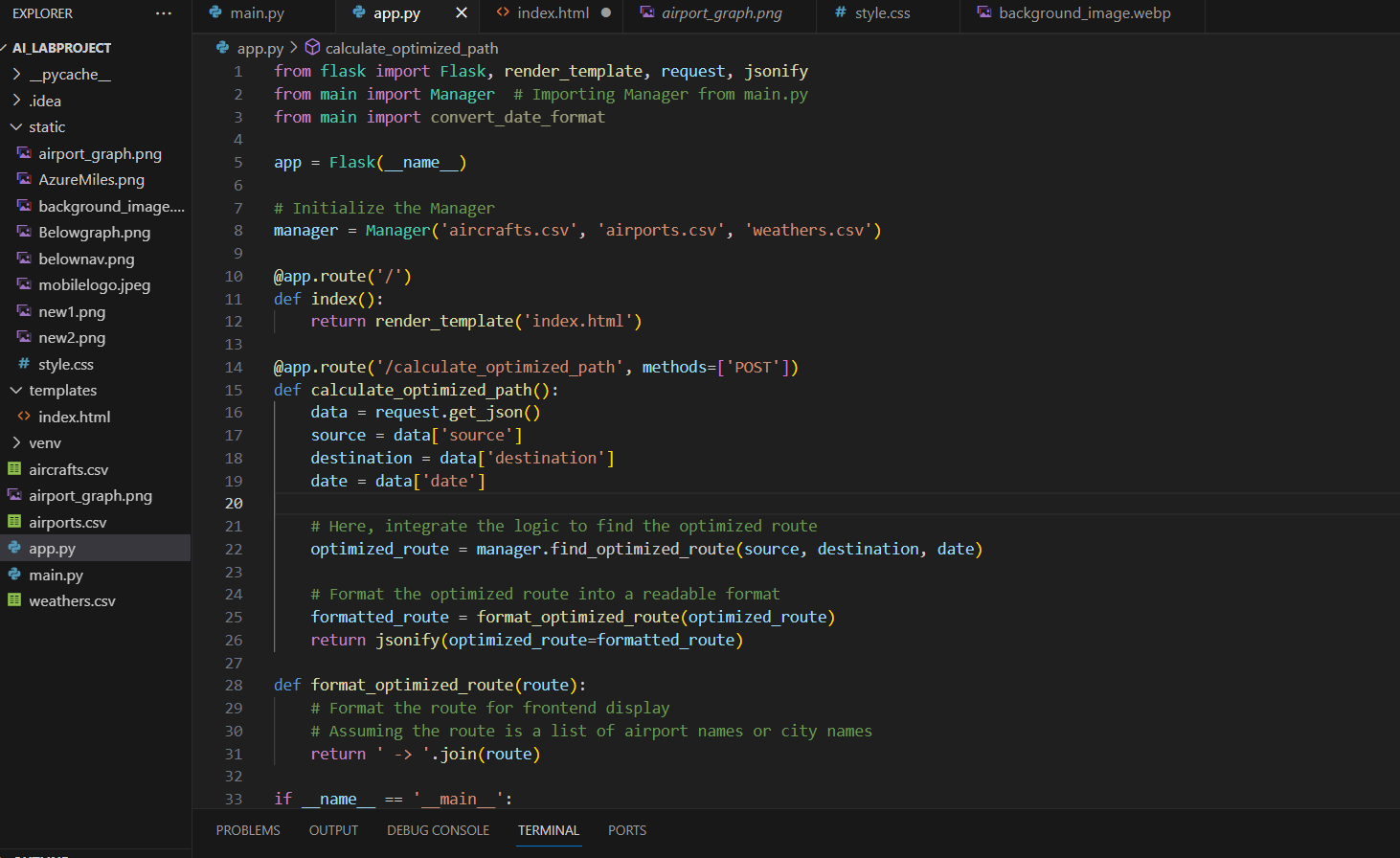
Description automatically generated

Similarly, a graph was also shown that shows the graphical interpretation of the backend logic and shows all the cities connected and the shortest path between them. On clicking the button “Show Airport Graph”, the following is printed on the screen:  


This image is being created by the backend which is then saved in the static folder and then directly used using the img src in the html.

Then comes the main output of the question:  
  
  
The user is prompted to enter the source City and then the destination city. Further for the date input the user is asked to select the date from the calender and then the search button calls the javascript in the end of the html file.

  
  
When a user clicks the "Search" button on the webpage, the HTML form captures their input for source, destination, and date. This input is then handled by JavaScript, which prevents the default form submission behavior and sends the data to the Flask backend using a POST request through the Fetch API. On the Flask backend, the calculate\_optimized\_path function processes this data, likely using complex algorithms or computations to find the optimized route. After computing, Flask sends the result back to the frontend. JavaScript then dynamically updates the webpage to display the optimized route, allowing for real-time feedback without needing to reload the page.

  
  
A screen shot of a computer code

Description automatically generated  
A screen shot of a computer program

Description automatically generated

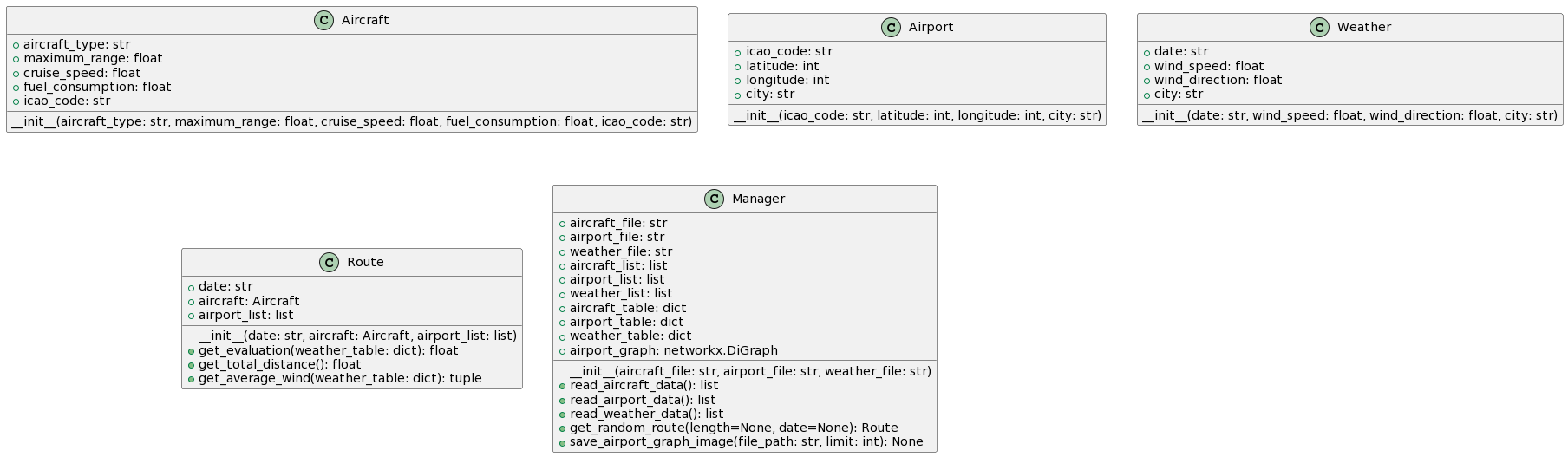
// @Rafay: add front end simulation logic explanation //

# **Route Optimization and Metrics**

The route optimization is handled by the route class. Each route consists of a starting airport, an ending airport, a list of intermediate airports, an aircraft and the date on which the flights will take place. Each route is evaluated on the basis of time taken and fuel consumed (equal weightage). The time taken is dependent on the total distance of the intermediate route, which is calculated on the basis of the sum of Euclidean distances between latitude and longitude of one airport and the next, and on the speed of the aircraft which is calculated using the aircraft’s given cruise speed and the effect of the average wind speed and average wind direction across the entire route on the flight. The fuel consumed is calculated using the aircraft’s fuel consumption at cruise speed and then factoring in the effect of wind.

The wind effect is such that if the wind is aligned with the aircraft’s direction, it will increase its speed but also increase fuel consumption and the opposite is true the more the wind goes against the aircraft’s direction.

# **Well Structured Code**



A class diagram of the design mentioned in the data pre-processing section

The code is divided into classes and functions, each performing its task as independently as possible. Optional arguments are provided where the client may or may not want to use them. The genetic algorithm code is structured properly and divided into its steps of selection, crossover, mutation, etc. The code is commented with proper explanation in most places. An example (of the Route class) is given below:

